Introduction to STATA

What STATA looks like

The STATA package is located under *Start -> Programs -> STATA*. You can also run the program by double-clicking a STATA data set (*.dta). STATA is a command-driven package. You type

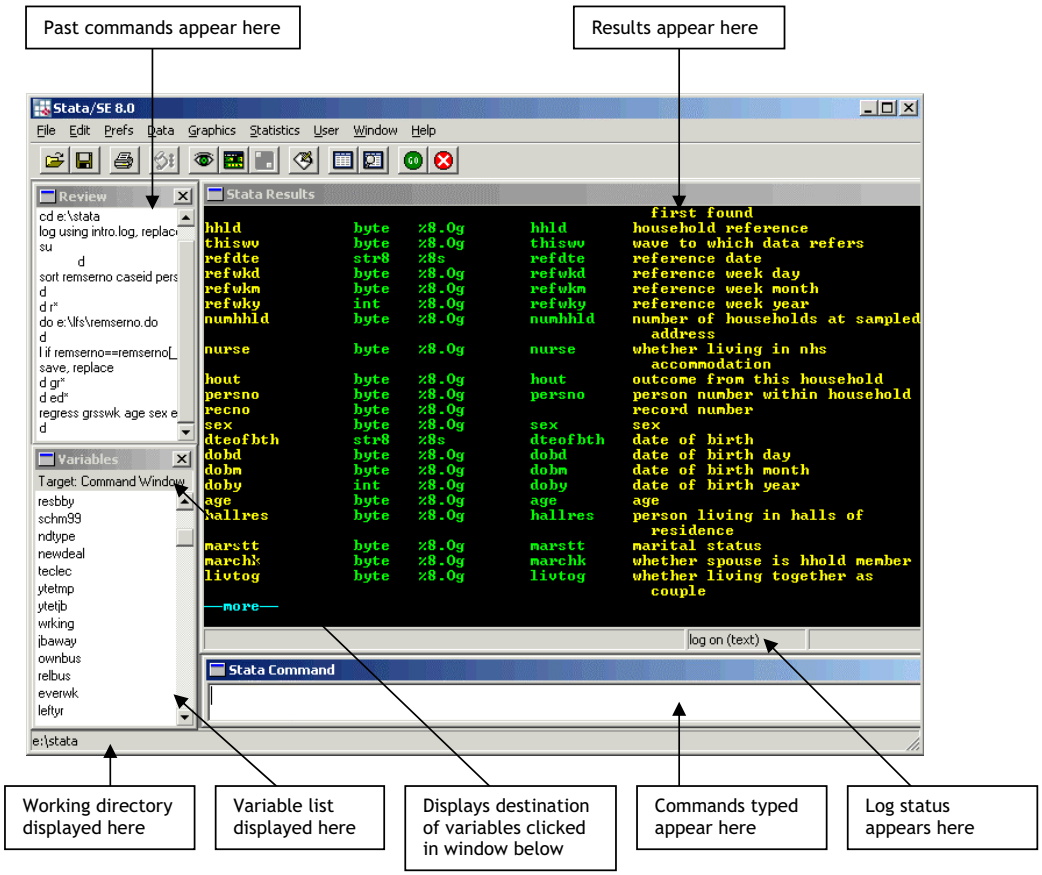

Figure 1: Screenshot

in the commands you want rather than pick them from pull-down Windows menus. It has many in-build commands that can make easy work of complicated statistical or econometric routines. Furthermore, you have the possibility to create your own commands to do anything that STATA is not equipped to do itself. You can enter commands in either of two ways:

- interactively: type the first command and execute it (hit *Return* or *Enter*), then the next, and so on.
- do-file: type up a list of commands in a 'do-file' (essentially a computer program) and execute the do-file.

Directories and folders

Like Dos and Windows, STATA can organise files in a tree-style directory with different folders. You can use this to organise your work in order to make it easier to find things at a later date. For

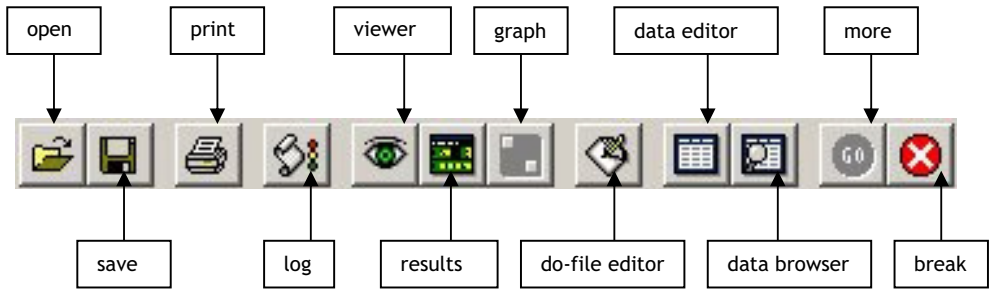

Figure 2: The STATA Toolbar

open	open a STATA dataset
save	save a dataset
print	print contents of active window
log	to start or stop, pause or resume a log file
viewer	open viewer window, or bring to the front
results	open results window, or bring to the front
graph	open graph window, or bring to the front
do-file editor	open do-file editor, or bring to the front
data editor	open data editor, or bring to the front
data browser	open data browser, or bring to the front
more	command to continue when paused in long output
break	stop the current task

Table 1: The STATA Toolbar

example, create a folder *data* to hold all the datasets you use, sub-folders, and so on. You can use some Dos commands in STATA, including: `cd, mkdir, dir`. Note, STATA is case sensitive, so it will not recognise the command `CD` or `Cd`. Also, quotes are only needed if the directory or folder names has spaces in it ("c:\temp\Stata folder") but it's a good habit to use them all time.

Getting help

- **Manuals:** The User Manual provides an overall view on using STATA. There are also a number of Reference Volumes, which are basically encyclopedias of all the different commands and all you ever needed to know about each one. If you want to find information on a particular command or a particular econometric technique, you should first look up the index at the back of any manual to find which volumes have the relevant information. Finally, there is a separate Graphics Manual.
- **Stata's in-built help and website:** STATA also has an abbreviated version of its manuals built-in. Click on *Help*, then *Contents*. STATA's website has a very useful FAQ section at

http://www.stata.com/support/faqs/. Both, the in-built help and the FAQs, can be simultaneously searched from within STATA itself (*Help*, then *Contents*). Helpful links are listed at http://www.stata.com/links/resources1.html/. Many researchers provide their own STATA programs on STATA's webpage.

```
.net search keyword
```

searches the internet for user-written additions to STATA that contain the specified *keyword*.

Reading data into STATA

There are different ways of reading or entering data into STATA:

- **use:** If your data is in STATA format, then simply read it in as follows:

  ```
  . use ``c:\auto.dta''
  ```

- **insheet:** If your data is originally in Excel or some other format, you need to prepare the data before reading it directly into STATA. You need to save the data in the other package (e.g. Excel) as either a *csv* (comma separated values) or *txt* (ASCII text) file. There are some rules to be followed when saving a *csv*- or *txt*-file for reading into STATA:
  - The first line in the spreadsheet should have the variables names, e.g. `make, price, mpg`, and the second line onwards should have the data. If the top row contains a title, delete this row.
  - Any extra line below the data or to the right of the data will also be read in by STATA, so make sure that only the data itself is in the spreadsheet. If necessary select all the bottom rows and/or right-hand columns and delete them.
  - Some notation for missing values can confuse STATA, e.g. it will read double dots (..) or hyphens (-) as text. Use find & replace to get symbols with single dots (.) or simply to delete them altogether.

  Once the *csv*- or *txt*-file is saved, you then read it into STATA using the command:

  ```
  . insheet using ``c:\auto.csv''
  ```

- The data in the active datafile can be browsed (read-only) in the *Browser* window, which is activated from the menu *Data/Data browser*, by clicking the browse-icon or by

  ```
  .browse varlist
  ```

  where the *varlist* is a list of variables to be displayed. The *Editor* window allows to edit data either by directly typing into the editor windor or by copying and pasting from spreadsheets software

  ```
  .edit varlist
  ```

Variable and data types

STATA supports six variable types, which are grouped into real numbers (*float, double*), integer (*byte, int, long*), and string (*#str*). The default type is *float*, single precision real number.

STATA stores or formats data in either of two ways — numeric or string. Numeric will store numbers while string will store text. Stings can also be used to store numbers, but you will not

be able to perform numerical analysis on those numbers. Note, with string variables, you must enclose the observation reference in double quotes. Otherwise, STATA will claim not to be able to find what you are referring to. For example:

```
. summarize if make==‘‘Dodge Colt’’
```

When writing commands, missing numeric observations are denoted by a single dot (.) and missing string observations are denoted by blank quotes (“ ”).

Examining the data
It is very important to examine your data when you first read it into STATA — you should check that all the variables and observations are read in as expected.

- **summarize:** This provides summary statistics, such as means, standard deviations, and so on.

  ```
  . summarize
  ```

  ```
  . sum

      Variable |       Obs        Mean    Std. Dev.       Min        Max
  -------------+--------------------------------------------------------
          make |         0
         price |        74    6165.257    2949.496      3291      15906
           mpg |        74     21.2973    5.785503        12         41
         rep78 |        69    3.405797    .9899323         1          5
      headroom |        74    2.993243    .8459948       1.5          5
  -------------+--------------------------------------------------------
         trunk |        74    13.75676    4.277404         5         23
        weight |        74    3019.459    777.1936      1760       4840
        length |        74    187.9324    22.26634       142        233
          turn |        74    39.64865    4.399354        31         51
  displacement |        74    197.2973    91.83722        79        425
  -------------+--------------------------------------------------------
    gear_ratio |        74    3.014865    .4562871      2.19       3.89
       foreign |        74    .2972973    .4601885         0          1
  ```

  Figure 3: Summarize

  This information is useful to double-check for problems with the data, e.g. if the min of some variable is negative this might indicate an error in the way the numbers were entered into STATA.

  As an aside, most STATA commands can be abbreviated, which saves some typing. Thus, for example, type `sum` instead of `summarize`. The abbreviations are noted in the STATA manuals.

- **graph:** You can graph a simple histogram with the command:

  ```
  . graph twoway histogram rep78
  ```

  This automatically calculates the dimensions of the histogram (e.g. number and width of bins/bars). You also can override the defaults)

  To draw a two-way scatterplot:

  ```
  . graph twoway scatter price mpg
  ```

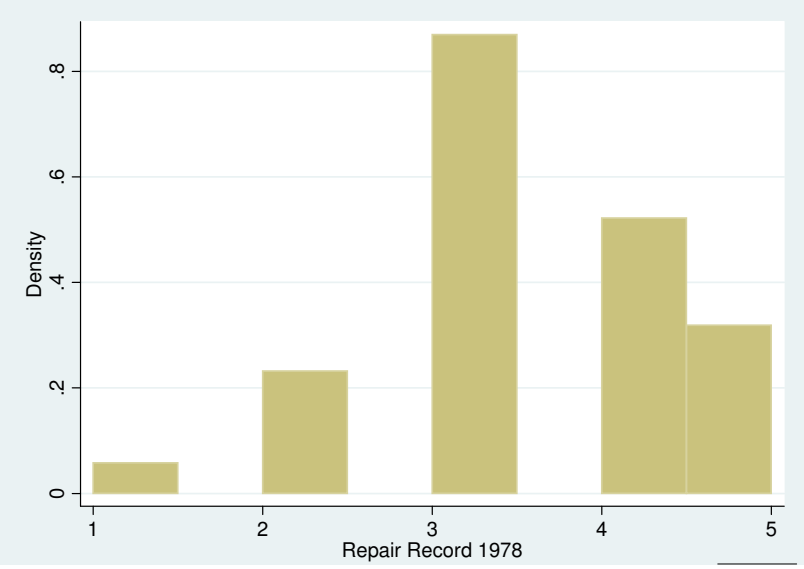

Figure 4: Histogram

Saving the dataset
The command is `save`, or `sa` for short:

```
. sa ‘‘c:\auto.dta’’, replace
```

The `replace` option overwrites any previous version of the file in the directory you try to saving to. If you want to keep an old version as back-up, you should save under a name, such as “auto_new”.

Preserve and restore
If you are going to make some revisions to the dataset but are unsure of whether or not you will keep them, then you have two options. First, you can save the current version, make the revisions, and if you decide not to keep them, just re-open the saved version. Second, you can use the `preserve` and `restore` commands; `preserve` will take a “photocopy” of the dataset as it stands and if you want to revert back to that copy later on, just type `restore`.

Organising datasets

- **rename:** You may want to change the names of your variables, perhaps to make it more transparent what the variable is:

  ```
  . ren mpg mileage

  . ren rep78 repair_record
  ```

  Note, you can only rename one variable at a time.

- **recode:** You can change the values that certain variables take, e.g. suppose that the trunk space is 13 instead of 11 in all possible cases:

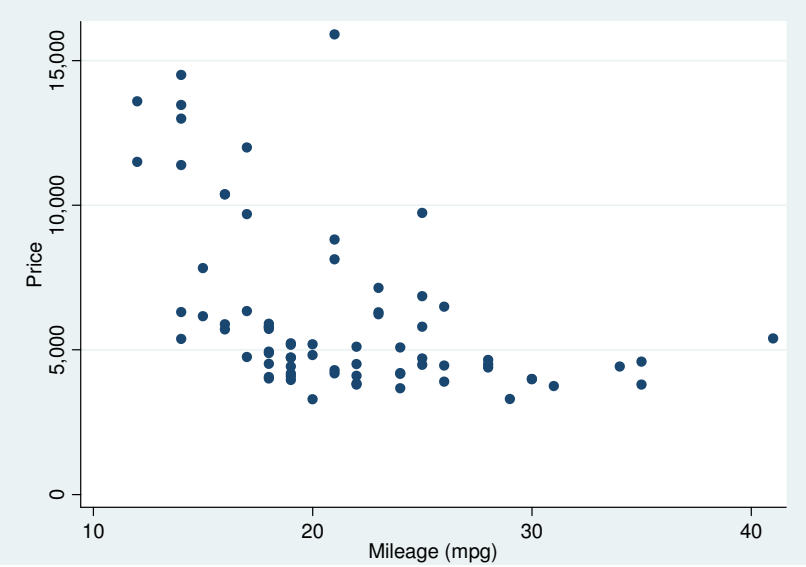

Figure 5: Scatterplot

```
. recode trunk (11=13)
```

Note, `recode` only allows numeric variables. This command can also be used to recode missing values to the dot that STATA uses to denote missings. Suppose a dataset codes missing `rep78` values as -999:

```
. recode rep78 -999 = .
```

- **keep and drop (including some notes on if-processing):** The original dataset may contain variables you are not interested in or observations you don't want to analyse. It's a good idea to get rid of these first — so they won't use valuable memory and these data won't inadvertently sneak into your analysis. You can tell STATA to either keep what you want or drop what you don't want — the end results will be the same.

```
. keep make price mpg rep78  weight foreign
. drop headroom trunk length turn displacement gear_ratio
```

Each of these will leave you with the same set of variables — you can double-check this in the variables window. You can also drop or keep observations, such as those with more than 3 repairs in 1978:

```
. keep if rep78 >= 3
. drop if rep78 < 3
```

Note, the different relational operators are:

==	equal to
~=	not equal to
>	greater than
>=	greater than or equal
<	less than
<=	less that or equal

Keeping observations only for the cars with repair records between 2 and 4:

```
. keep if rep78 >= 2 & rep78 <=4
. drop if rep78 < 2 | rep78 >4
```

Or keep that cars which have zero or 1, and 4 or 5 repairs in 1978:

```
. keep if (rep78 >= 0 & rep78 <2) | (rep78>3 & rep78<=5)
```

Note, the different logical operators are:

&	and
|	or
~ or !	not

You may want to drop observations with specific values, such as missing values (denoted in STATA by a dot):

```
. drop if rep78 ==.
```

- **sort:** For many STATA operations, it is important what order your observations are in. Thus, sometimes you have to `sort` the data:

  ```
  . sort price
  ```

  This puts the observations in ascending order (`browse` to check).

- **by-processing:** You can re-run a command for different subsets of data using the `by` prefix. For example, to get summary statistics of `price` broken down by `rep78`:

  ```
  . sort rep78
  . by rep78: sum price
  ```

  Note, for this operation you have to `sort` the data first. The `by` prefix causes the `sum` command to be repeated for each unique value of the variable `rep78`. The result is the same as writing a list of `sum` commands with separate `if` statements for each possible repair record:

  ```
  . sum price if rep78==1
  . sum price if rep78==2
  . sum price if rep78==3
  . sum price if rep78==4
  . sum price if rep78==5
  ```

  This example may not be very useful, but if you had a dataset for different countries, then you may want to run some commands for the different country subsets.

Creating new variables

- **System variables:** The following system variables (note the "_") may be useful:
  _n contains the number of the current observation.
  _N contains the total number of observations in the dataset.
  _pi contains the value of π to machine precision.
- **Generate, egen, replace:** The two most common commands for creating new variables are gen and egen.

  You can create a host of new variables from the existing data with the gen command (not all of the following examples have a senseful meaning):

```
. gen range = mpg * trunk / 1.609344  /* division, multiplication */
. gen var1 = headroom + length - 100  /* addition , subtraction */
. gen dif = abs(displacement - trunk) /* absolute difference */
. gen lprice = ln(price)              /* natural logarithm */
. gen weightsq = weight^2             /* square */
. gen cumprice = sum(price)           /* running cumulative */
. gen ten = 10                        /* constant value of 10 */
. gen id=_n                           /* id number of observation */
. gen small = mpg if price < 4000 & price ~= .
                                      /* mpg for small prices */
```

  Note, missing numeric observations, denoted by a dot, are interpreted by STATA as a very large positive number. You need to pay special attention to such observations when using if statements. If the last command listed above had simply been `gen small = mpg if price < 4000`, then small would have included any observation with missings. The egen command typically creates new variables based on summary measures, such as sum, mean, min, and max:

```
. egen totprice = sum(price), by(foreign)
. egen avgprice = mean(price), by(foreign)
. egen maxprice = max(price)
```

  Note, that both gen and egen have sum options. egen generates the total sum, and gen creates a cumulative sum. The running cumulation of gen depends on the order in which the data is sorted, so use with caution.

  The replace command modifies existing variables in exactly the same way as gen creates new variables:

```
.gen lprice = ln(price)
.replace lprice = 0 if lprice == .    /* missing now = 0 */
```

- **Dummy variables:** You can use gen and replace to create dummy variables as follows:

```
.gen smallprice = 0
.replace smallprice = 1 if price <= 4000 & price ~= .
```

  Or you can combine these in one command:

```
.gen smallprice = (price <= 4000 & price ~= .)
```

Note, the parenthesis are not strictly necessary, but can be useful for clarity purposes. If you want to create a set of dummy variables, for example, one for each rep78, use:

```
.tab rep78, gen(rdum)
```

This creates a dummy variable `rdum1` equal to one if the `rep78` variable is 1 (or whatever is the first entry in alphabetic order) and zero otherwise, a dummy variable `rdum2` if the entry is 2, and so on up to `rdum5`. You can refer to this set of dummies in later commands using a *wild card*, e. g., `rdum*`, instead of typing out the entire list.

- **Lags and leads:** Suppose that the variable `date` contains quarterly date information, say 19471, 19472, 19473, 19474 and then jumps to 19481. To generate a lagged `price` variable, we need our date variable to run continuously in regular increments. One way to solve this, would be to use the actual observation number (denoted `_n`), as they run continuously from 1, 2, 3 right up to 74 in single units. First, make sure to sort the data so that the dates so run in chronological order.

  ```
  .so date
  .gen period = _n
  .gen lagprice = price[_n-1] if period == period[_n-1] + 1
  ```

  This lags every observation by one quarter. The `if` argument avoids problems when there are gaps in the dates. You need to make sure that you are lagging the values of the previous period's data only – if the dataset only had observations for 19471 and 19501-20014, then lags will only be created for 19502 on.

  A lead can be created in similar fashion:

  ```
  .gen leadprice = price[_n+1] if period == period[_n+1]-1
  ```

Do-files and log-files

Instead of typing commands one-by-one interactively, you can type them all in one go within a *do-file* and simply run the *do-file*. The result of each command can be recorded in a *log-file* for review when th *do-file* is finished running. The vast majority of your work should use *do-files*. If you have a long list of commands, executing a *do-file* once is a lot quicker than executing several commands one after another. Furthermore, the *do-file* is a permanent record of all your commands and the order in which you ran them. This is useful if you need to "tweak" things or correct mistakes – instead of inputting all the commands again one after another, just modify the *do-file* and re-run it.

Do-files can be written in any text editor. STATA also has its own editor built in – click the icon along the top of the screen with the pad-and-pencil logo. Most *do-files* follows the following format:

```
version 8.0
clear
cd h:\projects\project1
capture log close
log using project1.log, replace
```

```
set more off
set memory 10m

LIST OF COMMANDS

log close
```

To explain the different commands:
`version` — STATA is backward compatible, so programs created using version 8 can still be run when version 9 comes along by just entering this command at the start.
`clear` — clears any data currently in STATA's memory.
`cd h:\projects\project1` — sets the default directory where STATA will look for any files you try to open and save any files you try to save.
`capture log close` — closes any *log-files* that you might have accidentally left open. If there were no *log-file* actually open, then the command `log close` on its own would stop the *do-file* running and give the error message: `no log file open`. Using `capture` tells STATA to ignore any error messages and keep going.
`log using project1.log, replace` — starts a *log-file* of all the results. The `replace` option overwrites any *log-file* of the same name. If, instead, you want to add the new *log-file* to the end of previous versions, then use the `append` option.
`set more off` — when there are a lot of results in the results window, STATA pauses the do-file to give you a chance to review each page on-screen and you have to press a key to get more. This command tells STATA to run the entire *do-file* without pausing. You can then review the results in the *log file*.
`set memory 10m` — STATA's default memory may not be big enough to handle large data files. Trying to open a file that is too large returns a long error message beginning: `no room to add more observations`. You can adjust the memory size to suit. Note, however, that setting it too large can take the PC's memory away from other applications and slow the computer down, so only set it as large as necessary.
`log close` — closes the log file. Remember to include a <Return> or <Enter> at the end of the last command to ensure that STATA executes it.

Comments
It is good practice to keep extensive notes within your *do-file* so that when you look back over it you know what you were trying to achieve with each command or set of commands. You can insert notes in two different ways:

- STATA will ignore a line if it starts with an asterisk *, so you can type whatever you like on that line. Note, the asterisk is also useful for getting STATA to temporarily ignore commands – if you decide later to re-insert the command into your do-file, just delete the asterisk.
- You can place notes after a command by inserting it inside these pseudo-parentheses, for example:

  ```
  .use ‘‘h:\auto.dta’’, clear   /* opens the auto data */
  ```

  These pseudo-parentheses are also useful for temporarily blocking a whole set of commands

– place /* at the beginning of the first command, */ at the end of the last, and STATA will just skip over them all.

Estimation

- **Linear estimation:** A straightforward OLS regression of mileage on weight, weight^2, and foreign gives the results presented in Figure 6:

```
.reg mpg weight weightsq foreign
```

```
. reg mpg weight weightsq foreign

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  3,    70) =   52.25
       Model |  1689.15372     3  563.05124             Prob > F      =  0.0000
    Residual |   754.30574    70  10.7757963            R-squared     =  0.6913
-------------+------------------------------           Adj R-squared =  0.6781
       Total |  2443.45946    73  33.4720474            Root MSE      =  3.2827

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |  -.0165729   .0039692    -4.18   0.000    -.0244892   -.0086567
    weightsq |   1.59e-06   6.25e-07     2.55   0.013     3.45e-07    2.84e-06
     foreign |     -2.2035   1.059246    -2.08   0.041      -4.3161   -.0909002
       _cons |    56.53884   6.197383     9.12   0.000     44.17855    68.89913
------------------------------------------------------------------------------
```

Figure 6: Regression

There are a few points to note here:

  - The first variable listed after the `regress` (or `reg` for short) command is the dependent variable, and all subsequently listed variables are the independent variables.
  - STATA automatically adds the constant term or intercept to the list of independent variables (type `reg ce gdp, noconstant` if you want to exclude it).
  - The top-left corner gives the ANOVA decomposition of the sum of squares in the dependent variable (Total) into the explained (Model) and unexplained (Residual).
  - The top-right corner gives the statistical significance results for the model as a whole, e.g. R-squared.
  - The bottom section gives the results for the individual independent variables, e.g. standard errors.
  - You can run regressions on a sub-sample using if-processing, e.g. `reg mpg weight weightsq foreign if price<=4000`.

- **Instrumental variable regression:** Theoretically, we could instrument the independent variable `weight` with other observed variables (cf. Figure 7):

```
.ivreg mpg foreign (weight = headroom trunk length)
```

Hypothesis testing

The results of each estimation automatically include for each independent variable a t-test (for linear regressions) and a z-test (for regressions such as logit or probit) on the null hypothesis that the "true" coefficient is equal to zero. You can also perform an F-test or χ^2-test on this hypothesis using the `test` command:

```
. ivreg mpg foreign (weight = headroom trunk length )

Instrumental variables (2SLS) regression

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  2,    71) =   66.82
       Model |  1614.56175     2  807.280876            Prob > F      =  0.0000
    Residual |  828.897707    71  11.6746156            R-squared     =  0.6608
-------------+------------------------------           Adj R-squared =  0.6512
       Total |  2443.45946    73  33.4720474            Root MSE      =  3.4168

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |  -.0069944   .0006951   -10.06   0.000    -.0083804   -.0056084
     foreign |  -2.057039   1.113337    -1.85   0.069    -4.276969    .1628918
       _cons |   43.02817   2.353938    18.28   0.000     38.33456    47.72179
------------------------------------------------------------------------------
Instrumented:  weight
Instruments:   foreign headroom trunk length
------------------------------------------------------------------------------
```

Figure 7: IV regression

```
.ivreg mpg foreign (weight = headroom trunk length)
.test weight = 0
```

or, since STATA defaults to comparing the listed terms to zero, you can simply use:

```
.test weight
```

The F-statistic (cf. Figure 8) with 1 numerator and 71 denominator degrees of freedom is 101.25. The p-value or significance level of the test is zero, so we can reject the null hypothesis even at the 1% level – `weight` is significantly different from zero. Notice, that the F-distribution with 1 numerator degree of freedom is identical to the t^2-distribution, so the F-test result is the same as the square of the t-test result in the regression. Also the p-values associated with each test agree.

If you want to, you can test any linear hypothesis about th coefficients, such as:

```
.reg mpg weight weightsq foreign
.test 3 * weight = 0.5 * weightsq - 2 * foreign
```

If you want to jointly test a number of restrictions, use the `accumulate` option:

```
.test weight = 0
.test foreign = 3, accum
```

or you can simply use

```
.test (weight=0) (foreign=3)
```

```
. test weight

 ( 1)  weight = 0

       F(  1,    71) =  101.25
            Prob > F =    0.0000
```

Figure 8: F-test